**Практика №8**

**Создание простого HTTP-сервера с помощью Java. Фреймворк Spring. Создание веб-приложения.**

## Теоретическое введение

HTTP (протокол передачи гипертекста) — это основной протокол взаимодействия в сети интернет, позволяющий клиентам (например, браузерам) общаться с серверами. HTTP использует модель запрос-ответ: клиент отправляет запрос, а сервер отвечает. Рассмотрим основные компоненты HTTP-сервера.

* **TCP-сокеты**: HTTP протокол функционирует на основе TCP-соединений. Для работы с сокетами в Java используется класс ServerSocket.
* **HTTP-запросы**: HTTP-запрос состоит из метода (например, GET, POST), пути (URL) и HTTP-заголовков.
* **HTTP-ответы**: Сервер должен отправить клиенту ответ, который включает код статуса (200, 404 и т.д.), HTTP-заголовки и тело ответа (например, HTML).

Java предоставляет широкие возможности для реализации HTTP-серверов. Ниже будет приведен листинг HTTP-сервера на Java. HTTP-сервер будет реализовывать функционал заметок. На сервере будет реализованы функции добавления заметок, удаления последней заметки, а также производится проверка на удаление несуществующей заметки. Пример кода приведен на Листинге 15.1.

### Листинг 15.1 – Пример реализации HTTP-сервера на Java для работы с заметками

java

import java.io.\*;

import java.net.\*;

import java.util.\*;

public class NoteHttpServer {

private static final int PORT = 8080;

private static final List<String> notes = new ArrayList<>();

public static void main(String[] args) {

try (ServerSocket serverSocket = new ServerSocket(PORT)) {

System.out.println("Note HTTP Server запущен на порту " + PORT);

while (true) {

try (Socket clientSocket = serverSocket.accept();

PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()))) {

String inputLine;

StringBuilder request = new StringBuilder();

while ((inputLine = in.readLine()) != null && !inputLine.isEmpty()) {

request.append(inputLine).append("\n");

}

String[] requestLines = request.toString().split("\n");

String method = requestLines[0].split(" ")[0];

String path = requestLines[0].split(" ")[1];

String response;

if (method.equals("GET") && path.equals("/notes")) {

response = "HTTP/1.1 200 OK\r\n" +

"Content-Type: text/html\r\n" +

"\r\n" +

"<html><body><h1>Notes</h1><ul>";

for (String note : notes) {

response += "<li>" + note + "</li>";

}

response += "</ul></body></html>";

} else if (method.equals("POST") && path.equals("/add")) {

String note = requestLines[requestLines.length - 1].trim();

if (!note.isEmpty()) {

notes.add(note);

response = "HTTP/1.1 200 OK\r\n" +

"Content-Type: text/html\r\n" +

"\r\n" +

"<html><body><h1>Note added</h1></body></html>";

} else {

response = "HTTP/1.1 400 Bad Request\r\n" +

"Content-Type: text/html\r\n" +

"\r\n" +

"<html><body><h1>Empty note</h1></body></html>";

}

} else if (method.equals("POST") && path.equals("/remove")) {

if (!notes.isEmpty()) {

notes.remove(notes.size() - 1);

response = "HTTP/1.1 200 OK\r\n" +

"Content-Type: text/html\r\n" +

"\r\n" +

"<html><body><h1>Last note removed</h1></body></html>";

} else {

response = "HTTP/1.1 400 Bad Request\r\n" +

"Content-Type: text/html\r\n" +

"\r\n" +

"<html><body><h1>No notes to remove</h1></body></html>";

}

} else {

response = "HTTP/1.1 404 Not Found\r\n" +

"Content-Type: text/html\r\n" +

"\r\n" +

"<html><body><h1>404 Not Found</h1></body></html>";

}

out.print(response);

} catch (IOException e) {

e.printStackTrace();

}

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

## Практическое задание

В данной практической работе представлено 4 варианта заданий. Выбор варианта осуществляется в соответствии с последней цифрой номера студента в списке. Если последняя цифра номера студента равна 0 или 1, выполняется задание варианта №1; если последняя цифра равна 2 или 3 — выполняется вариант №2; если последняя цифра равна 4 или 5 — выполняется вариант №3; если последняя цифра равна 6, 7, 8 или 9 — выполняется вариант №4.

### Вариант №1

Расширьте функционал HTTP-сервера из Листинга 15.1, добавив возможность редактирования существующей заметки по её индексу. Например, запрос POST /edit?index=0&text=NewText должен изменить заметку с индексом 0 на "NewText". Обработайте случаи некорректного индекса или пустого текста.

### Вариант №2

Добавьте в HTTP-сервер из Листинга 15.1 поддержку статических файлов (например, CSS или изображения). Реализуйте обработку запросов типа GET /style.css, где сервер будет возвращать содержимое файла из определённой директории (например, public/).

### Вариант №3

Модифицируйте HTTP-сервер из Листинга 15.1, добавив авторизацию. Реализуйте базовую проверку логина и пароля (например, через заголовок Authorization). Доступ к заметкам должен быть ограничен, если пользователь не авторизован.

### Вариант №4

Расширьте HTTP-сервер из Листинга 15.1, добавив поддержку метода DELETE для удаления заметки по индексу (например, DELETE /notes?index=0). Обработайте случаи некорректного индекса или отсутствия заметки с указанным индексом.

## Теоретическое введение.

Spring Framework — это один из самых популярных фреймворков для разработки корпоративных приложений на языке Java. Он предоставляет мощные инструменты для упрощения разработки, управления зависимостями, обработки данных и интеграции с различными технологиями. Spring активно используется для создания веб-приложений, микросервисов и других типов программного обеспечения.

### Основные принципы Spring

* **Инверсия управления (IoC)** Spring использует контейнер Inversion of Control для управления зависимостями между компонентами приложения. Вместо того чтобы создавать объекты и управлять их зависимостями вручную, Spring автоматически настраивает и связывает компоненты.
* **Внедрение зависимостей (Dependency Injection, DI)** DI позволяет передавать зависимости через конструкторы, сеттеры или поля, снижая связность компонентов и упрощая их тестирование.
* **Модульность и расширяемость** Spring — это набор модулей (Core, Data Access, Web, Security и др.), что позволяет использовать только те компоненты, которые нужны для проекта.
* **Легковесность** Несмотря на широкий функционал, Spring остается относительно легковесным, с минимальными затратами на инициализацию и управление ресурсами.

### Ключевые модули Spring Framework

* **Spring Core** Основной модуль, включающий ядро фреймворка, контейнер IoC и DI. Он обеспечивает управление жизненным циклом объектов и их зависимостей.
* **Spring AOP (Aspect-Oriented Programming)** Позволяет разделять бизнес-логику и вспомогательные операции (например, логирование, транзакции), используя аспекты.
* **Spring MVC (Model-View-Controller)** Модуль для создания веб-приложений, предоставляющий архитектуру MVC для обработки HTTP-запросов и рендеринга страниц.
* **Spring Data** Упрощает доступ к данным, предоставляя абстракции для работы с базами данных (JPA, JDBC) и другими источниками данных.
* **Spring Security** Модуль для обеспечения безопасности приложений, включая аутентификацию и авторизацию.
* **Spring Boot** Упрощает настройку и запуск приложений Spring, предоставляя встроенный сервер и автоматическую конфигурацию.

## Практическое задание

В данной практической работе представлено 4 варианта заданий. Выбор варианта осуществляется в соответствии с последней цифрой номера студента в списке. Если последняя цифра номера студента равна 0 или 1, выполняется задание варианта №1; если последняя цифра равна 2 или 3 — выполняется вариант №2; если последняя цифра равна 4 или 5 — выполняется вариант №3; если последняя цифра равна 6, 7, 8 или 9 — выполняется вариант №4.

### Вариант №1

Создайте простое веб-приложение с использованием Spring Boot, которое реализует REST API для управления списком задач (To-Do List). Требования:

* Создать сущность Task с полями id, title и completed.
* Реализовать контроллер с методами:
  + GET /tasks — возвращает список всех задач.
  + POST /tasks — добавляет новую задачу.
  + DELETE /tasks/{id} — удаляет задачу по идентификатору.
* Использовать аннотации @RestController и @RequestMapping.

### Вариант №2

Разработайте веб-приложение с использованием Spring MVC, которое позволяет пользователям добавлять и отображать комментарии. Требования:

* Создать сущность Comment с полями id, author и text.
* Реализовать контроллер с методами для отображения формы добавления комментария и сохранения комментария в список.
* Использовать шаблонизатор Thymeleaf для рендеринга HTML-страниц.
* Отображать список всех комментариев на главной странице.

### Вариант №3

Создайте веб-приложение с использованием Spring Boot и Spring Data JPA, которое управляет библиотекой книг. Требования:

* Создать сущность Book с полями id, title, author и isbn.
* Настроить подключение к H2 в памяти как базу данных.
* Реализовать репозиторий с методами для добавления, поиска и удаления книг.
* Создать REST API с методами:
  + GET /books — возвращает список всех книг.
  + POST /books — добавляет новую книгу.

### Вариант №4

Разработайте веб-приложение с использованием Spring Boot и Spring Security для аутентификации пользователей. Требования:

* Настроить базовую аутентификацию с использованием username и password.
* Создать контроллер с защищённым методом /welcome, доступным только авторизованным пользователям.
* Реализовать страницу логина с использованием Thymeleaf.
* Добавить простую роль USER и проверить её в конфигурации безопасности.